# CONSIST–Consistent Internet Route Updates *

Tania Mishra and Sartaj Sahni
Department of Computer and Information Science and Engineering,
University of Florida, Gainesville, FL 32611
{tmishra, sahni}@cise.ufl.edu

November 10, 2009

**Abstract**

An Internet router may receive a batch of tens of thousands of BGP (Border Gateway Protocol) updates (insert a new rule or delete/change an existing rule) in any instant (i.e., with the same timestamp). This paper deals with analyzing possible orderings of a batch of updates such that forwarding table consistency is maintained while these updates are performed one at a time as in a table that supports incremental updates rather than batch updates. This analysis results in the system CONSIST, which removes any redundancies in the batch of updates and uses a heuristic to arrange the reduced set of updates into a consistent sequence that results in near minimal increase in table size as the updates are done one by one.

**Keywords**
Internet packet forwarding, route updates, table consistency, incremental updates

## 1 Introduction

The Internet is a large collection of autonomous systems (AS) around the world, where the autonomous systems are interconnected by the huge Internet backbone network comprising of high capacity data routes and core routers. An autonomous system (AS) is in itself a collection of networks and routers under a single administrative control. Examples of AS include networks in companies, universities and ISPs. As a packet moves from source to destination, it passes a number of routers in its path. Each router forwards the packet to the appropriate output from which the packet resumes its journey to the next router. Routers at the border of an AS exchange reachability information (for prefixes within the AS and those outside the AS and reachable through it) with border routers of other ASes. The Internet uses BGP for inter-AS routing.

BGP is essentially an incremental protocol in which a router generates update messages only when there is a change in its routing state. Such a change could be in the network topology (for example when a router fails or comes up after failure or is added to the network) or in the routing policy. The BGP update messages are sent over semi-permanent TCP connections to the neighboring routers. Two routers connected by TCP are called BGP peers. To forward a packet from input to an appropriate output, each router maintains a forwarding table that consists of rules of the form $(P, H)$, where $P$ is a prefix and $H$ is the next hop corresponding to $P$. Forwarding rules are inserted, deleted or changed on a continuous basis in response to the BGP updates. In our paper, we use functions "$insert(P, H)$", "$delete(P)$" and "$change(P, H)$" to represent the insertion, deletion and change respectively of a rule in the forwarding table. A BGP update message can be one of two types: an announcement or a withdrawal

A route announcement advertises a new route to a prefix. Upon receiving an announcement, a router compares the newly advertised route with the existing ones for the same prefix in the routing information base (RIB or routing table). If there are no existing routes then a new prefix is added to the forwarding table along with the next hop as IP address of the BGP peer from which the announcement was received. If, on the other hand, there

*This research was supported, in part, by the National Science Foundation, under grant 0829916.

are existing routes, then the new route is compared with the existing ones by applying the BGP selection rules. If the new route is superior to the best among the existing routes, then the rule corresponding to the route is changed in the forwarding table by changing the next hop to point to the BGP peer that sent the new route announcement. If the new route is inferior to the best existing route, then the announcement has no effect on the forwarding table and hence the routing policy. The new route is stored in the RIB in any case.

A route withdrawal message, similarly, triggers a number of actions at a router. The router removes the route from the RIB, and then checks if there are existing routes from different peers to the same prefix. If there are no such routes, then the forwarding rule for the route is deleted from the forwarding table. On the other hand, if there are more routes and the withdrawn route was the best among them, then the next best route is picked from the remaining routes by applying the BGP selection rules and the forwarding table is updated by changing appropriately the nexthop of the rule corresponding to the route. Otherwise, if the withdrawn route is not the best among the existing routes, then the forwarding table is left unchanged.

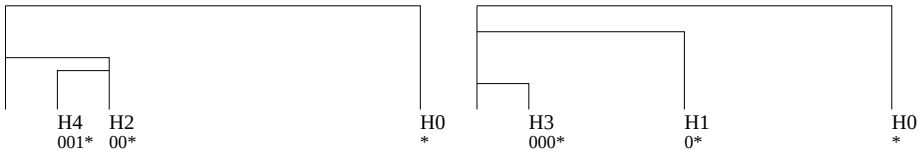
The route announcements and withdrawals arrive at a router in groups instead of one at a time. Some of the reasons are:

1. Multiple BGP messages are packed into a single message to minimize the number of route advertisements [5]. In this case, multiple announcements and withdrawals are received in one message from a BGP peer.

2. The network could get self-synchronized [29, 22], which is a state where a large number of BGP routers may transmit updates simultaneously. In this case multiple update messages are received at about the same time from different BGP peers.

3. the BGP update messages are received at a router with an intermediate delay [27] where the delay can be caused by Minimum Route Advertisement Interval (MRAI, [6]), TCP buffering or fragmentation during packet transmission or reception, by transient problems affecting multihop sessions, or by the CPU remaining busy. Due to this intermediate delay, multiple messages could accumulate and be ready for processing at the same time.

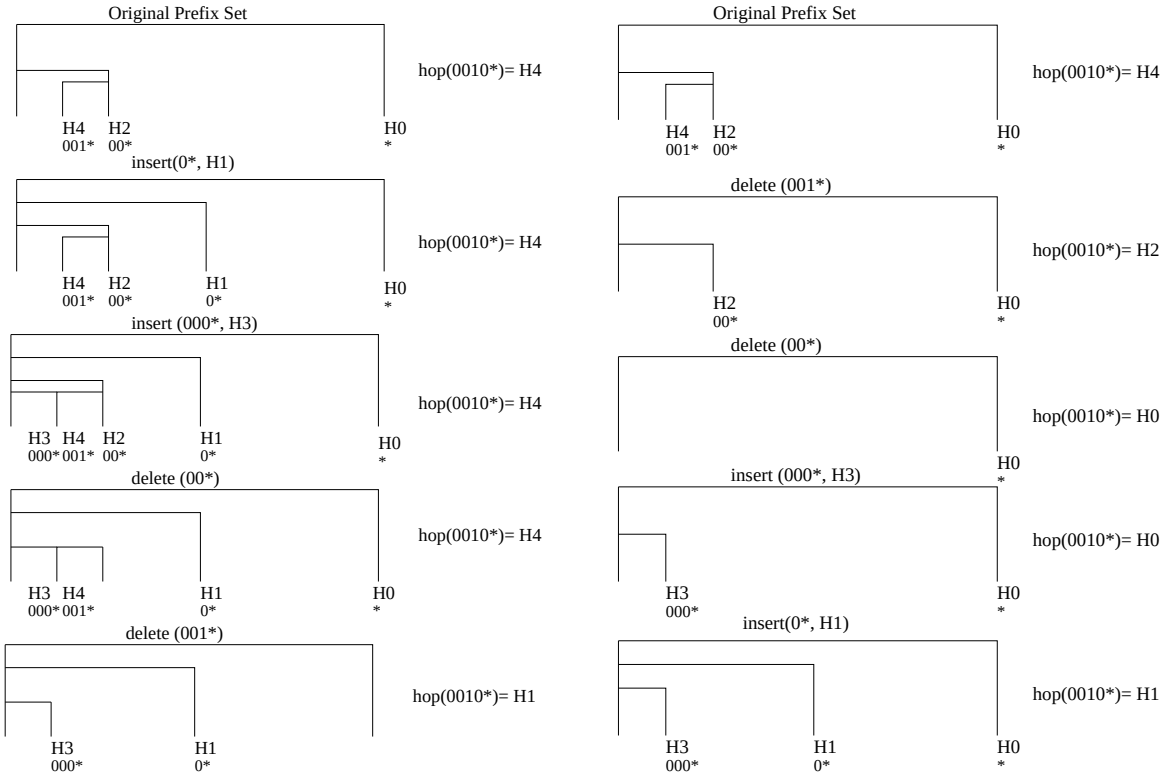
Since BGP routing protocols take time in the order of minutes to converge [18], the forwarding table need not change more than once a second and indeed from the update traces at [3, 4], we observe that BGP update messages are timestamped to the precision of a second, with a number of messages having the same timestamp. At the peak rate, there are tens of thousands of such messages registered with the same timestamp. However, most router tables perform updates one at a time (i.e., incrementally) in the control plane concurrent with lookup operations in the data plane. By incrementally performing the updates in an update batch in a carefully selected order, it is possible for an incrementally updated router to behave exactly like one that is batch updated. Note that in a batch updated router, packets are routed to a next hop determined either by the table state before any update is done or by the state following the incorporation of all updates in an update batch.




(a) before applying updates hop(0010*)=H4 (b) after applying updates hop(0010*)=H1

Figure 1: Prefixes in forwarding table before and after applying updates

Informally, a forwarding table is consistent when every lookup returns a next hop that would be returned just before or after a batch of updates is applied. For example, suppose a forwarding table contains rules: $(001*, H4)$, $(00*, H2)$, $(*, H0)$ as shown in Figure 1(a) and the following updates are received in a batch: $delete(001*)$, $delete(00*)$, $insert(000*, H3)$, $insert(0*, H1)$. Figure 1(b) gives the prefixes in the forwarding table after the updates are applied. Now, suppose a data packet with destination address beginning with 0010 is received. If no update has been processed yet, then from Figure 1(a) nexthop $H4$ is returned. If on the other hand, the batch of updates is completely processed, then from Figure 1(b) the returned nexthop is $H1$. Thus, as we apply the updates we must ensure that the packets are forwarded to either $H1$ or $H4$. For example, if we apply the updates in the




(a) batch consistent updates with hops on the side after each update

(b) batch consistency violated, with hops H0 and H2 being selected

Figure 2: Prefixes in forwarding table as updates are being applied in two different sequences

following sequence: $insert(0*, H1)$, $insert(000*, H3)$, $delete(00*)$, $delete(001*)$, then consistency is maintained at every step as we see from Figure 2(a). On the other hand, if we apply the updates as they come, then consistency is not maintained as we see in Figure 2(b), where the returned hops is one from: $\{H0, H1, H2, H4\}$, at any time. This example deals with batch consistency, which we will define later.

The main contribution of this paper is to formalize the consistency properties of incrementally updated forwarding tables in the face of arrival of updates in batches. We define and analyze requirements for two types of consistency, namely, batch consistency and incremental consistency. We represent the sequence of updates using precedence graphs to ensure batch consistency, and give a heuristic to obtain a near optimal batch consistent sequence as a topological ordering of vertices of the precedence graph. Here, optimality is defined with respect to the increase in size of the intermediate forwarding table, where an optimal sequence guarantees minimum increase in the maximum table size. We also show that it is not possible to construct a precedence graph representation for all incremental consistent sequences.

We begin with Section 2 by reviewing related work. In Section 3 we present the consistency requirements and analysis. Our consistency management system that results from the development of Section 3 is described in Section 4. An experimental evaluation of the update traces at various routers for processing updates in batches and eliminating redundancies is given in Section 5 and we conclude in Section 6.

# 2 Related Work

With the Internet being under a constant flux of change, there has been a significant amount of work in the recent past dealing with how to incorporate the changes to the forwarding table. The two broad categories of work in this area include one in which the updates are applied incrementally [12, 13, 15, 14, 8, 17] and the other in which

the updates are applied in batches [19, 10, 11].

To apply the updates in batches, typically a duplicate copy of the forwarding table is maintained so that when one copy is involved in lookup the other can be updated and after the update is complete, the lookups are done in the updated copy. Batch updates require double the memory footprint, however it is useful for situations when incremental updates are compute-intensive. Batch updates have been used in high-performance router design in [20]. Incremental updates, on the other hand, are applied to the same copy of the forwarding table that is used for lookups. Update and lookup operations could use time multiplexing to share the common memory that stores the prefix database [12, 13, 14]. Thus the database is locked when the update operations are carried out, to prevent a lookup operation from taking place at the same time.

Z. Wang et. al in CoPTUA [8] developed an approach of performing updates without locking the database (stored in a TCAM) from lookups while ensuring that the lookups return consistent and non-erroneous results. They define a *consistent rule table* to be a rule table in which the rule matched (including the action associated with the rule) by a look up operation performed in the data plane is either the rule (including action) that would be matched just before or just after any ongoing update operation in the control plane. Wang et al. [8] develop a scheme for consistent table update without locking the TCAM at any time, essentially allowing a search to proceed while the table is being updated [17]. Other TCAM designs that support consistent incremental updates on a per operation basis without locking for lookups are [16, 17].

When we look at the actual update traces we find that the updates are received at a router in groups with the same timestamp. A group could contain upto tens of thousands of updates arriving at the same time. Applying the updates in the same order as their arrival sequence could result in consistency issues, namely, that shorter prefixes could get matched during lookup resulting in a non-optimal forwarding for a number of packets. Both [8] and [16] suggest handling a batch of updates by doing the delete first and then the inserts. This strategy, however, does not ensure table consistency. Also, in their cases, the batch of updates get generated at a router while processing a single update to the forwarding or filter table. Thus, when the updates are scheduled in a batch, they do not consider the other BGP updates that have been received and are ready to be applied at the same time.

We next show how the batch of updates are generated in [16] and [8]. The authors in [16] maintain a leaf-pushed prefix trie. As a result, any insert, delete or change operation to a non-leaf trie node could cause multiple leaf prefixes to be changed, deleted or inserted. In fact, in the worst case, when there is an operation on the default prefix (to be applied to the root node of the trie) there could be $O(n)$ extra operations [17]. So, the scope of consistency in this work is limited to the multiple update operations generated hence. The authors in [8] show how multiple policy filter rules get affected by the introduction or deletion of a filter rule and how in the worst case $N_r/2$ rule moves may be required for updating a filter table consisting of $N_r$ rules. The scope of consistency is limited to each rule move resulting from a rule addition or deletion. When CoPTUA [8] is applied to a forwarding table using the incremental update algorithms found in [13], the consistency criteria is then applied to each rule move (of a maximum of 16 for IPv4) in the forwarding table. In our work, we broaden the scope of consistency by recognizing all the updates received since the last BGP update was processed, and by arranging the updates in a sequence that maintains consistency of the forwarding table as the updates are applied incrementally.

Despite measures (such as route-flap damping) adopted in routers, redundancies in update messages still exist. The authors in [24] reduce the number of writes to the prefix trie by suppressing the "redundant" writes. A redundant write is one that is succeeded by another write and after the succeeding write is applied, the state of the prefix database is the same irrespective of whether the redundant write was applied. In our work here, we offer a detailed algorithm for computing a reduction of a given update sequence based on the "*insert*", "*delete*" and "*change*" functions.

# 3 Consistent Updates

## 3.1 Definitions and Properties

We begin with definitions for batch and incremental consistent sequences.

**Definition 1** *Let $U = u_1, \cdots, u_r$ be an update sequence; each $u_i$ is an insert, delete, or change operation. Let $T_0$ be a router table and let $T_i(U)$ be the state of this router table after the operations $u_1, \cdots, u_i$ have been performed, in this order, on $T_0$. Let $T_0(U) - T_0$ and let $nextHop(d, R))$ be the next hop associated with the longest matching*

*prefix* $LMP(d, R)$ *for the destination address* $d$ *in router table* $R$. *Let* $S = s_1, \cdots, s_m$ *be another update sequence.* $S$ *is* **batch consistent** *with respect to* $T_0$ *and* $U$ *iff*

$$T_r(U) = T_m(S) \wedge \forall i \forall d[nextHop(d, T_i(S)] \in \{nextHop(d, T_0), nextHop(d, T_r(U))\}$$

$S$ *is* **incremental consistent** *with respect to* $T_0$ *and* $U$ *iff*

$$T_r(U) = T_m(S) \wedge \forall i \forall d[nextHop(d, T_i(S)] \in \{nextHop(d, T_0), \cdots, nextHop(d, T_r(U))\}$$

Note that two tables are equal iff they contain exactly the same rules. Further, although $U$ is always incremental consistent with respect to itself, it is generally not batch consistent with respect to itself (i.e., $S = U$) and a table $T_0$. For example, when $U = insert(1*, H1), delete(1*, H1)$ $T_0 = \{(*, H0)\}$ and $H0 \neq H1$, $T_2(U) = T_0$ $nextHop(d, T_0) = nextHop(d, T_2) = H0$ for all $d$. However, even though $T_r(U) = T_m(S)$, $nextHop(d, T_1(S)) = H1 \neq H0$ for every $d$ that begins with 1.

**Definition 2** *The update sequence* $V(U)$ *(or simply* $V$*) derived from the update sequence* $U$ *in the following manner is called the* **reduction** *of* $U$.

*Examine the update operations in the order* $u_1, u_2, \cdots$.
*Let* $P$ *be the prefix associated with the operation* $u_i$ *being examined.*
*If* $P$ *occurs next in* $u_j$, $j > i$, *do the following:*

1. *If* $u_i = change(P, newHop)$ *and* $u_j = change(P, newHop')$, *remove* $u_i$ *from* $U$. *If* $newHop'$ *is the same as the existing hop for* $P$ *in the forwarding table, then remove* $u_j$ *from* $U$ *as well.*
2. *If* $u_i = change(P, newHop)$ *and* $u_j = delete(P)$, *remove* $u_i$ *from* $U$
3. *If* $u_i = delete(P)$ *and* $u_j = insert(P, hop)$, *remove* $u_i$ *from* $U$ *and replace* $u_j$ *by* $change(P, hop)$. *($u_j$ may also be removed from $U$ when hop equals the current hop associated with $P$ in the router table.)*
4. *If* $u_i = insert(P, hop)$ *and* $u_j = change(P, newHop)$, *remove* $u_i$ *from* $U$ *and replace* $u_j$ *by* $insert(P, newHop)$.
5. *If* $u_i = insert(P, hop)$ *and* $u_j = delete(P)$, *remove* $u_i$ *and* $u_j$ *from* $U$.

*Note that the remaining four possibilities for* $u_i$ *and* $u_j$ *((change, insert), (delete, change), (delete, delete), and (insert, insert)) are invalid.*

For example, the reduction of $U = insert(1*, H1), insert(01*, H2), delete(1*, H1)$ is $V = insert(01*, H2)$. It is easy to see that a prefix $P$ may be associated with at most one operation in the reduction of $U$.

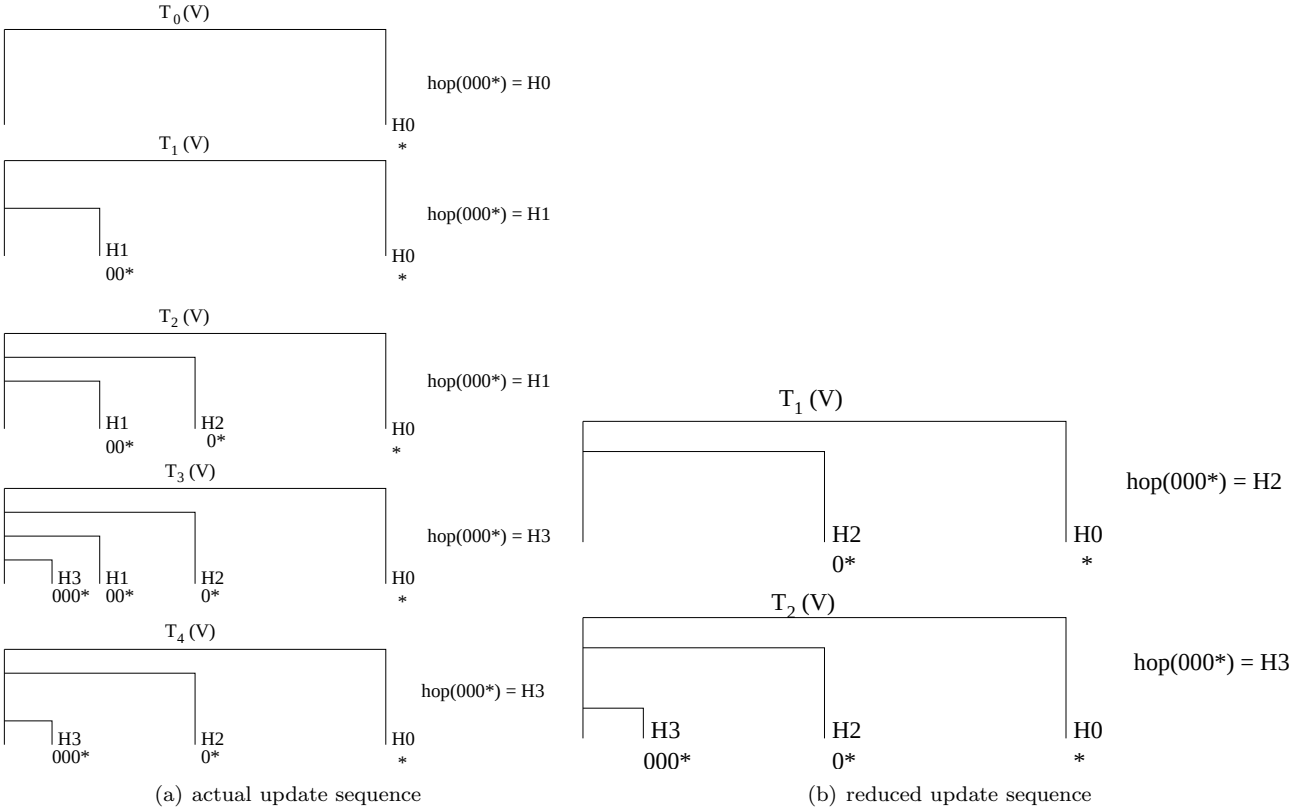
We note that the reduced update sequence $V(U)$ may be neither batch nor incremental consistent with respect to $U$ and $T_0$. To see this, suppose that $T_0 = \{*, H0\}$ and $U = insert(00*, H1), insert(0*, H2), insert(000*, H3), delete(00*)$. Figure 3(a) shows $T_0$, $T_1(U)$, $\cdots$, $T_4(U)$. Batch consistency requires the next hops for destination addresses matched by 000* to be either $H0$ or $H3$ while incremental consistency requires these next hops to be $H0$, $H1$ or $H3$. Figure 3(b) shows $T_1(V)$ and $T_2(V)$ for the reduced sequence $V(U) = insert(0*, H2), insert(000*, H3)$. We see that $nextHop(d, T_1(V)) = H2$ for addresses $d$ matched by $000*$. So, $V(U)$ is neither batch nor incremental consistent with respect to $U$ and $T_0$.

Theorem 1 establishes the existence of a batch consistent update sequence for every router table $T_0$ and every update sequence $U$. Note that the existence of an incremental consistent update sequence follows from our earlier observation that $U$ is incremental consistent with respect to itself and every $T_0$. This follows also from Theorem 1 as every batch consistent update sequence is incremental consistent as well. An incremental consistent sequence may not, however, be batch consistent.

**Theorem 1** *For every router table* $T_0$ *and update sequence* $U$, *there exists a batch consistent update sequence* $S$.

**Proof** Let $S = s_1, \cdots, s_m$ be derived from $U = u_1, \cdots, u_r$ as below.

**Step 1** Let $V = v_1, \cdots, v_m$ be the reduction of $U$.




(a) actual update sequence

(b) reduced update sequence

Figure 3: Actual and reduced update sequences

**Step 2** Reorder the operations of $V$ so that the inserts are at the front and in decreasing order of prefix length; followed by the change operations in any order; followed by the deletes in increasing order of prefix length. Call the resulting sequence $S$.

We shall show that $S$ is batch consistent with respect to $U$ and every $T_0$. First, it is easy to see that $T_r(U) = T_m(V) = T_m(S)$. So, we need only to show

$$\forall i \forall d[nextHop(d, T_i(S)] \in \{nextHop(d, T_0), nextHop(d, T_m(S))\}$$

The proof is by induction on $i$. For the induction base, $i = 0$, $T_0(S) = T_0$, and so $\forall d[nextHop(d, T_0(S) = nextHop(d, T_0)]$. For the induction hypothesis, assume that

$$\forall d[nextHop(d, T_j(S) \in \{nextHop(d, T_0), nextHop(d, T_m(S))\}]$$

for some $j$, $0 \leq j < m$. In the induction step, we show

$$\forall d[nextHop(d, T_{j+1}(S) \in \{nextHop(d, T_0), nextHop(d, T_m(S))\}] \quad (1)$$

If $\forall d[nextHop(d, T_{j+1}(S) = nextHop(d, T_j(S))]$, then Equation 1 follows from the induction hypothesis. So, suppose there is a $d$ such that $nextHop(d, T_{j+1}(S) \neq nextHop(d, T_j(S))$. There are three cases to consider for $s_{j+1}$– $insert(P, hop)$, $change(P, newHop)$, and $delete(P)$. When $s_{j+1} = insert(P, hop)$ or $s_{j+1} = change(P, newHop)$, it must be that $LMP(d, T_{j+1}(S) = P$. Because of the ordering of operations in $S$, the remaining operations $s_{j+2}, \cdots$ do not change either the longest matching prefix for $d$ or the next hop associated with this prefix. So, $LMP(d, T_m(S)) = P$ and $nextHop(d, T_{j+1}(S) = nextHop(d, T_m(S))$ (see Figure 4). When $s_{j+1} = delete(P)$, it must be that $LMP(d, T_{j+1}(S) = P'$, where $P'$ is the longest prefix in $T_{j+1}(S)$ that encloses $P$ in $T_j(S)$. Because of the ordering of operations in $S$, the remaining operations $s_{j+2}, \cdots$ do not change either the longest

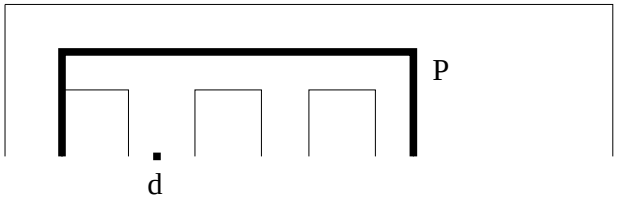



Figure 4: $T_{j+1}$. $P$ is newly inserted/deleted/changed prefix

matching prefix for $d$ or the next hop associated with this prefix. So, $LMP(d, T_m(S)) = LMP(d, T_j(S))$ and $nextHop(d, T_{j+1}(S) = nextHop(d, T_m(S))$. ■

From the proof of Theorem 1, we see that for an update sequence to be batch consistent, it is necessary only that whenever an operation changes the next hop for a destination $d$, that change be to the next hop in the final table $T_r(U)$. Using this observation, it is possible to construct additional batch consistent update sequences. For example, we could partition the operations in the reduction of U so that the addresses covered by the prefixes in one partition are disjoint from those covered by other partitions. Then, for each partition, we can order the operations as in the construction of Theorem 1 and concatenate these orderings to obtain a batch consistent update sequence. Different batch consistent update sequences may result in intermediate router tables of different size. As an example, consider the reduced update sequence of Figure 5. The outermost prefix and all prefixes marked D are in $T_0$. Those marked I are prefixes that are to be inserted and those marked D are to be deleted. The batch consistent sequence constructed in the proof of Theorem 1 will do all the inserts first and then the deletes. If there are $a$ inserts, the table size increases by $a$ following the last insert and then decreases back to the size of $T_0$ as the deletes complete. For the update sequence to succeed (when inserts/deletes are done incrementally, i.e., one at a time), we must have $a$ units of additional table capacity. An alternative batch consistent update sequence follows each insert with the deletion of its enclosed prefix that is labeled D in the figure. Using this sequence, only 1 additional unit of table capacity is required and this is optimal for the given example as it isn't possibly to do a delete before its enclosing insert and maintain batch consistency (unless, of course, the table is locked for lookup from the start of a delete to the completion of its enclosing insert).

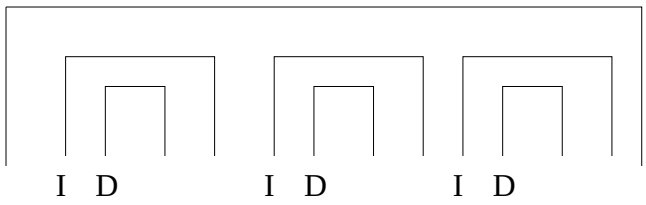



Figure 5: A bad example for sequence of Theorem 1

**Theorem 2** *For every router table $T_0$ and update sequence $U = u_1, \cdots, u_r$, the reduced sequence $V(U)$ has the smallest number of operations needed to transform $T_0$ to $T_r(U)$.*

**Proof** Consider any sequence $S$ that transforms $T_0$ to $T_r(U)$. Let $V(S)$ be the reduction of $S$. Clearly, $|V(S)| \leq |S|$ and $V(S)$ also transforms $T_0$ to $T_r$. We shall show that $|V(U)| \leq |V(S)|$, thereby proving the theorem.

Consider any $v_i \in V(U)$. If $v_i = insert(P, hop)$, then $P \notin T_0$ (follows from the correctness of $U$ with respect to $T_0$ and the definition of $V(U)$) and $P \in T_r(U)$. Consequently, $insert(P, hop') \in V(S)$. Since $P$ appears only once in $V(S)$, $hop' = hop$. So, $v_i \in (V(S)$. If $v_i = delete(P)$, then $P \in T_0$ and $P \notin T_r(U)$. The only way for $V(S)$ to accomplish this transformation is if $delete(P) \in V(S)$. Similarly, when $v_i = change(P.newHop)$, $v_i \in V(S)$. So, $V(U) \subseteq V(S)$ and $|V(U) \leq |V(S)|$. ■

**Definition 3** *A batch (incremental) consistent sequence $S = s_1, \cdots, s_m$ for $U$ and $T_0$ is* **optimal** *iff it minimizes $\max_{0 \leq i \leq m}\{|T_i(S)|\}$ relative to all batch (incremental) consistent sequences.*

**Theorem 3** *Let $S$ be an optimal batch consistent sequence for $U$ and $T_0$. Let $optB(T_0, U) = \max_{0 \leq i \leq m}\{|T_i(S)|\}$. Let $optI(T_0, U)$ be the corresponding quantity for an optimal incremental consistent sequence. $optB(T_0, U) - m/2 \leq optI(T_0, U) \leq optB(T_0, U)$ and both upper and lower bounds on $optI(T_0, U)$ are tight.*

**Proof** $optI(T_0, U) \leq optB(T_0, U)$ follows from the observation that every batch consistent sequence also is incremental consistent. To see that this is a tight upper bound, consider the case when $U$ is comprised only of change (or only of insert or only delete) operations. Now, $optI(T_0, U) = optB(T_0, U)$.

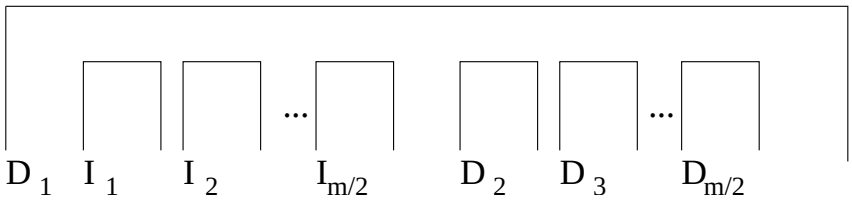



Figure 6: An example for sequence of Theorem 3

To establish $optB(T_0, U) - m/2 \leq optI(T_0, U)$, we note that $optI(T_0, U) \geq |T_0| + \max\{0, \#inserts(U) - \#deletes(U)\}$ and $optB(T_0, U) \leq |T_0| + \#inserts(U)$. So, $optB(T_0, U) - optI(T_0, U)$ is maximum when $\#inserts(U) = \#deletes(U)$. Since, $\#inserts(U) + \#deletes(U) \leq m$, $optB(T_0, U) - optI(T_0, U)$ is maximum when $\#inserts(U) = \#deletes(U) = m/2$. At this time, $optB(T_0, U) - optI(T_0, U) = m/2$. Hence, $optB(T_0, U) - m/2 \leq optI(T_0, U)$. For the tightness of this bound, consider a sequence $U$ of $m/2$ deletes followed by $m/2$ inserts as in Figure 6 (one delete that encloses m/2 inserts and m/2-1 deletes all of which are independent). Since $U$ is incremental consistent with itself, $optI(T_0, U) = |T_0|$. Batch consistency limits us to permutations of $U$ in which the inserts precede all the deletes. So, $optB(T_0, U) = |T_0| + m/2$. Hence, $optB(T_0, U) - m/2 = optI(T_0, U)$. ■

## 3.2 Batch Consistent Sequences

### 3.2.1 Graph Representation of Updates

When performing the updates $U = u_1, \cdots, u_r$ in a batch consistent manner, our primary objective is to perform the fewest possible inserts/deletes/changes to transform $T_0$ to $T_r$ and our secondary objective is to perform these fewest updates in a batch consistent order that minimizes the maximum size of an intermediate table. The primary objective is met by using the reduction $V(U)$ of $U$ (Theorem 2). For the secondary objective, we construct an $m$ vertex digraph $G(V)$ from $V = v_1, \cdots, v_m$. Vertex $i$ of $G$ represents the update operation $v_i$. Let $P_i$ be the prefix associated with update $v_i$, $1 \leq i \leq m$. There is a directed edge between vertices $i$ and $j$ iff (a) $P_i$ is a prefix of $P_j$, (b) there is no $P_k$ such that $P_i$ is a prefix of $P_k$ and $P_k$ is a prefix of $P_j$, and (c) one of the following relationships between $v_i$ and $v_j$ hold.

1. $v_i$ and $v_j$ are inserts $\Rightarrow (j, i) \in E(G)$, where $E(G)$ is the set of directed edges of $G$.
2. $v_i$ is an insert and $v_j$ is a delete $\Rightarrow (i, j) \in E(G)$.
3. $v_i$ and $v_j$ are deletes $\Rightarrow (i, j) \in E(G)$.
4. $v_i$ is a delete and $v_j$ is an insert $\Rightarrow (j, i) \in E(G)$.
5. $v_i$ is a change and $v_j$ is an insert $\Rightarrow (j, i) \in E(G)$.
6. $v_i$ is a change and $v_j$ is a delete $\Rightarrow (i, j) \in E(G)$.

**Definition 4** *$i$ is an* immediate predecessor *of $j$ in $G(V)$ iff $(i, j) \in E(G)$. $i$ is a* predecessor *of $j$ iff there is a directed path from $i$ to $j$ in $G(V)$.*

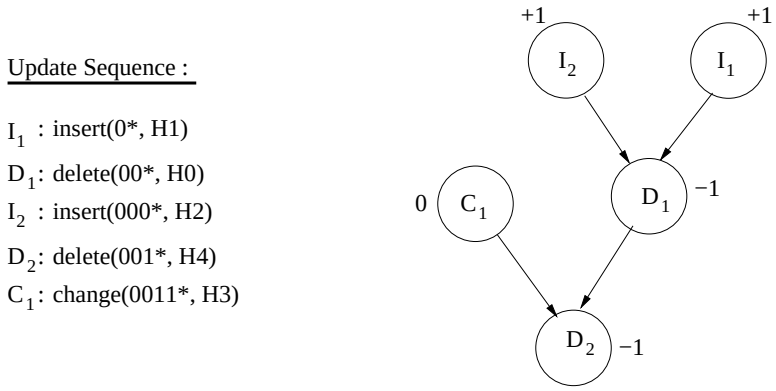



Figure 7: A reduced update sequence and its digraph

### 3.2.2 Near-optimal batch consistent sequence

We assign a weight of 1, 0, or $-1$ to vertex $i$ of $G(V)$ depending on whether $v_i$ is an insert, change, or delete. Figure 7 gives an example reduced update set $V$ and its corresponding digraph $V(G)$. One may verify that a permutation of a reduced update set $V$ is batch consistent iff it corresponds to a topological ordering of the vertices of $G(V)$. Further, for every topological ordering, $|T_i| - |T_0|$ equals the sum of the weights of the first $i$ vertices in the ordering.

**Definition 5** *For a given topological order, we define $w_i$ to be the sum of the weights of the first $i$ vertices. The* max weight *of a topological order is* $\max\{w_i\}$. *An* optimal *topological ordering is a topological ordering that has minimum max weight.*

Notice that our secondary objective is met by an optimal topological ordering. Although we have been unable to develop an efficient algorithm to find an optimal topological order, we propose an efficient heuristic (Figure 8) that does this for all tested router-table update traces. Our heuristic constructs a topological order in several rounds. In each round, one of the remaining deletes is selected to be the next delete in the topological order being constructed. In case no delete remains in $G(V)$, any topological ordering of the remaining vertices may be concatenated to the ordering so far constructed to complete the overall topological order. Assume at least one delete remains in $G(V)$. Only deletes that remain in $G(V)$ and that have no delete predecessors are candidates for the next delete. Each candidate delete $d$ is assigned an $(a, b)$ value where $a$ is the number of insert predecessors of $d$ and $b = a$ - the number of delete successors of $d$ (including $d$) that have no insert predecessors that are not also predecessors of $d$. From the candidate deletes, we select one using the following rule.

1. If the least $b$ is less than 0, from among the candidate deletes that have negative $b$, select one with least $a$.
2. If the least $b$ equals 0, select any one of the candidate deletes that have $b = 0$.
3. If the least $b$ is more than 0, from among the candidate deletes, select one with largest $a - b$.

Once the next delete for the topological ordering is selected, we concatenate its remaining predecessor inserts and changes (these inserts and changes are first put into topological order) to the topological ordering being constructed followed by the selected delete $d$ followed (in topological order) by the delete successors (and the remaining change predecessors of these delete successors) of $d$ that have no remaining insert predecessors. All newly added vertices to the topological ordering being constructed (together with incident edges) are deleted from $G(V)$ before commencing the next round selection. Our heuristic is motivated by the following two theorems.

**Theorem 4** *For every $G(V)$, there exists an optimal topological ordering in which between any two successive deletes $d_i$ and $d_{i+1}$ we have only the predecessor inserts and changes of $d_{i+1}$ that are not predecessor inserts and changes of any of the deletes $d_1, \cdots, d_i$. Here $d_1, \cdots$ are the deletes of $V$ indexed in the order they appear in the topological ordering.*

**Algorithm: computeOrder(G(V))**
create *candidateDeletes* from G(V), by listing all the delete vertices in G(V) that have no predecessor deletes.
**for** each delete vertex $d \in$ *candidateDeletes*
  compute (a($d$), b($d$)) where
  a($d$) = number of insert predecessors of $d$;
  b($d$) = a - 1 - number of delete successors of $d$ without insert predecessors;
**endfor**
nextDelete = NULL;
**while** there is a delete in *candidateDeletes*
  **if** there is only one delete $d$ in *candidateDeletes*
    nextDelete = $d$;
  **else**
    b_smallest = smallest b among b's for all deletes in *candidateDeletes*.
    **if** b_smallest $< 0$ **then**
      nextDelete = $d \in$ *candidateDeletes* s.t. b($d$)<0 and a($d$) is smallest
    **else if** b_smallest == 0 **then**
      nextDelete = $d \in$ *candidateDeletes* s.t. b($d$)==0
    **else**
      nextDelete = $d \in$ *candidateDeletes* s.t. a($d$)-b($d$) is largest
    **endif**
  **endif**
  append(nextDelete); // append to topological order and delete appended vertices from G(V).
  **if** any new deletes are found whose predecessor deletes have been appended and hence they have no remaining predecessor deletes **then**
    compute (a, b) values for all such freed deletes and add them to *candidateDeletes*.
  **endif**
**endwhile**
concatenate the remaining vertices in any topological ordering

Figure 8: Algorithm to compute a near-optimal topological order

**Proof** Consider an optimal topological ordering of $G(V)$. Examine the deletes left to right. Let $d_{i+1}$ be the first delete such that there is an insert or change between $d_i$ and $d_{i+1}$ that is not a predecessor of $d_{i+1}$. All inserts and changes between $d_i$ and $d_{i+1}$ that are not predecessors of $d_{i+1}$ may be moved from their present location in the topological ordering (without changing their relative ordering) to just after $d_{i+1}$. This relocation of the inserts and changes yields a new topological ordering that also is optimal. Repeating this transformation a finite number of times results in an optimal topological ordering that satisfies the theorem. ■

For the second theorem that motivates our heuristic, let $Q = \{(a_1, b_1), (a_2, b_2), \cdots, (a_n, b_n)\}$ be a set of pairs of numbers. We introduce the following notation:

1. $\sigma(Q)$ is a permutation of the pairs of $Q$ and $\sigma(Q, i)$ is the $i$th pair in this permutation. For simplicity, $\sigma(Q)$ and $\sigma(Q, i)$ will be abbreviated to $\sigma$ and $\sigma(i)$.

2. $B(\sigma(Q), i) = \sum_{j=1}^{i} b_{\sigma(j)}$. $B(\sigma((Q), i)$ will be abbreviated to $B(i)$. Note that $B(i)$ is the sum of the second coordinates of the first $i$ pairs of $\sigma$.

3. $A(\sigma(Q), i) = \max_{1 \le j \le i}\{B(j-1) + a_{\sigma(j)}\}$ and is abbreviated $A(i)$.

$\sigma(Q)$ is an *optimal permutation* iff it minimizes $A(n)$.

**Theorem 5** *Let $\sigma(Q)$ be such that:*

1. *The pairs with negative bs come first followed by those with zero bs followed by those with positive bs.*
2. *The pairs with negative bs are in increasing (non-decreasing) order of as.*
3. *The pairs with zero bs are in any order.*
4. *The pairs with positive bs are in decreasing (non-increasing) order of $a - b$.*

*$\sigma(Q)$ is an optimal sequence.*

**Proof** First, we show that permutations that violate one of the listed conditions cannot have a smaller $A(n)$ than those that satisfy all conditions. Consider a permutation that does not satisfy the conditions of the theorem. Suppose that the first violation of these conditions is at position $i$ of the permutation (i.e., pairs $i$ and $i+1$ of the permutation violate one of the conditions). Let $(a_1, b_1)$ be the $i$th pair and $(a_2, b_2)$ be the $i+1$st pair. Let $\Delta = \max\{a_1, b_1 + a_2\}$ and $\Delta' = \max\{a_2, b_2 + a_1\}$. We shall show that $\Delta' \leq \Delta$. This together with the observation that $A(i+1) = \max\{A(i-1), B(i-1) + a_1, B(i-1) + b_1 + a_2\}$ imply that swapping the pairs $i$ and $i+1$ does not increase $A(i+1)$. By repeatedly performing these violation swaps a finite number of times, we obtain a permutation that satisfies the conditions of the theorem and that has an $A(n)$ value no larger than that of the original permutation. Hence, a permutation that violates a listed condition cannot have a smaller $A(n)$ than one that satisfies all conditions.

To show $\Delta' \leq \Delta$, we consider the four possible cases for a violation of the conditions of the theorem– (a) $b_1 \geq 0$ and $b_2 < 0$ (violation of condition 1), (b) $b_1 > 0$ and $b_2 = 0$ (violation of condition 1), (c) $a_1 > a_2$, $b_1 < 0$, and $b_2 < 0$ (violation of condition 2), and (d) $a_1 - b_1 < a_2 - b_2$, $b_1 > 0$, and $b_2 > 0$ (violation of condition 4). Note that condition 3 cannot be violated as this condition permits arbitrary ordering of pairs with zero $b$. In fact, we see that when $b_1 = b_2 = 0$, $\Delta = \Delta' = \max\{a_1, a_2\}$ and swapping the pairs $i$ and $i+1$ does not affect $A(i+1)$.

*Case (a):* We see that $b_1 + a_2 \geq a_2$ and $b_2 + a_1 < a_1$. So, $\Delta' \leq \Delta$.

*Case (b):* Now, $\Delta' = \max\{a_2, b_2 + a_1\} = \max\{a_2, a_1\} \leq \Delta$.

*Case (c):* Now, $\Delta' < a_1 = \Delta$.

*Case (d):* From $a_1 - b_1 < a_2 - b_2$, it follows that $a_1 + b_2 < b_1 + a_2$. From this and $b_2 > 0$, we get $a_1 < b_1 + a_2$. Hence, $\Delta = b_1 + a_2$. If $a_2 \geq b_2 + a_1$, $\Delta' = a_2 < \Delta$. If $a_2 < b_2 + a_1$, $\Delta' = b_2 + a_1 < b_1 + a_2 = \Delta$.

To complete the proof, we need to show that all $\sigma$s that satisfy the conditions of the theorem have the same value of $A(n)$. Specifically, we need to show that $\Delta = \Delta'$ whenever (c') $a_1 = a_2$, $b_1 < 0$, and $b_2 < 0$ (tie in condition 2), and (d') $a_1 - b_1 = a_2 - b_2$, $b_1 > 0$, and $b_2 > 0$ (tie in condition 4). This is easily established. ■

Since the vertex weights in $G(V)$ are 1, 0, and $-1$, the max weight of a topological ordering cannot exceed $m$, the number of vertices in $V(G)$ and cannot be less than $-1$. The maximum occurs, for example, when all $v_i$ are inserts and the minimum happens, for example, when all $v_i$ are deletes. So, a topological ordering may have a max weight that exceeds the minimum max weight by $O(m)$. Our heuristic may produce topological orderings whose max weight is $\Omega(m)$ more than that of the optimal ordering. For example, consider the digraph of Figure 9 that has two components. The first component is comprised of a delete $d_1$ that has $m/3-2$ inserts that are immediate predecessors and $m/3-4$ immediate successor deletes. The second component has a delete $d_2$ that has 2 immediate predecessor inserts and a successor delete $d_3$ that also has 2 immediate predecessor inserts and $m/3-1$ immediate successor deletes. Deletes $d_1$ and $d_2$ are the candidate deletes during the first round of our heuristic. Their $(a, b)$ values are $(m/3-2, -1)$ and (2,1), respectively. $d_1$ is selected by our heuristic and the partial topological ordering constructed has $m/3-2$ inserts followed by $m/3-3$ deletes. In the next round $d_2$ preceded by its 2 predecessor inserts is added to the ordering. Finally, in the third round, $d_3$ preceded by its 2 predecessor inserts and followed by its $m/3-1$ successor deletes is added. The max weight of the constructed topological ordering is $m/3-2$ (assume $m/3 \geq 4$). In an optimal ordering, the first component appears after the second and the max weight is 3. So, the heuristic ordering has a max weight that is $m/3-5 = O(m)$ more than optimal.

Whenever each component of $V(G)$ is a *delete star* as in Figure 10, our heuristic finds an optimal ordering. Note that in a delete star, we have a delete vertex all of whose predecessors are inserts and/or changes and all of whose successors are deletes that have no additional predecessor inserts. This follows from Theorem 5 and the observation that each component has only one delete that ever becomes a candidate for selection by our heuristic. In general, whenever no component of $V(G)$ has two deletes that become candidates for selection, our heuristic obtains an optimal topological ordering. Although this property doesn't hold for the $V(G)$s that arise in practice, the $V(G)$s that arise in practice have a sufficiently simple structure that our heuristic obtains optimal topological orderings. Figure 11 shows some of the more complex components in the $V(G)$s of trace update data.

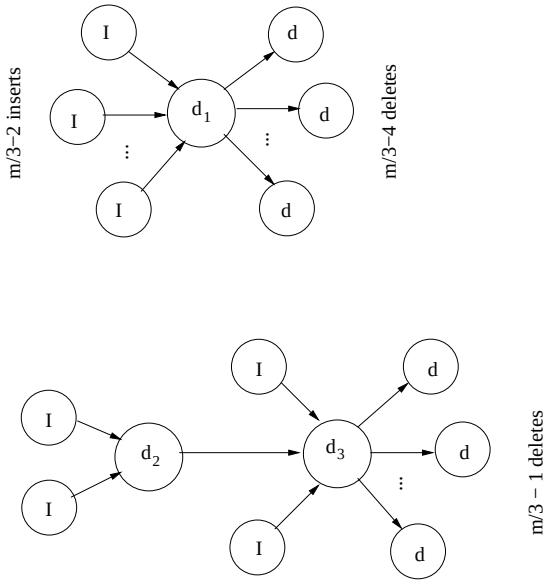



Figure 9: A digraph for which our heuristic produces sub-optimal ordering

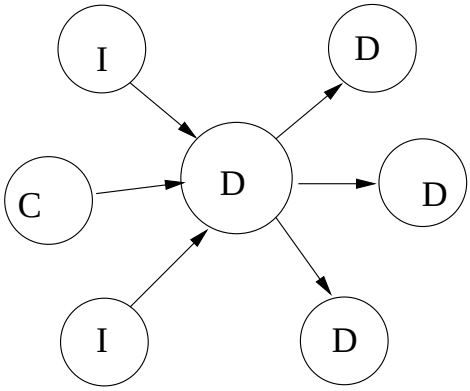



Figure 10: A delete star

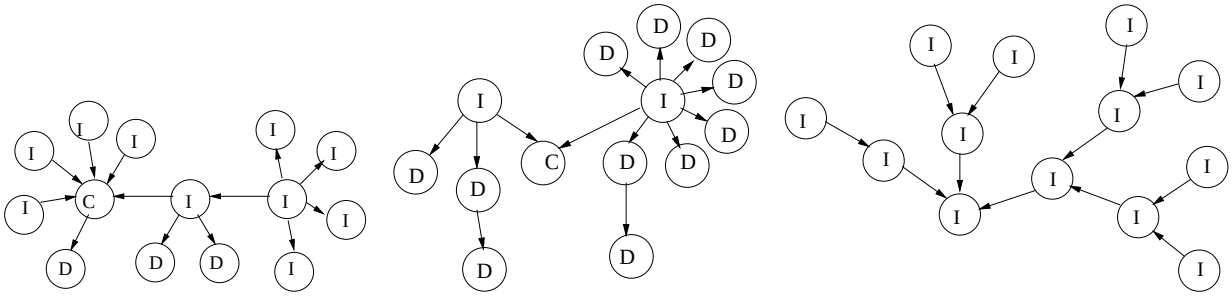


Figure 11: Some of the complex digraph components of the trace update data.

### 3.3 Incremental Consistent Sequences

When performing the updates $U = u_1, \cdots, u_r$ in an incremental consistent manner, the primary and secondary objectives are the same as those for batch consistency. That is, the primary objective is to perform the fewest

possible inserts/deletes/changes to transform $T_0$ to $T_r$ and the secondary objective is to perform these fewest updates in an incremental consistent order that minimizes the maximum size of an intermediate table. The primary objective is met by using the reduction $V(U)$ of $U$ (Theorem 2). Note that since $V(U)$ has a batch consistent ordering, it has also an incremental consistent ordering. For the secondary objective, however, there is no digraph $H(V)$ whose topological orderings correspond to the permissible incremental consistent orderings of $V(U)$. To see this, consider $T_0 = \{(*, H0), (00*, H1)\}$ and $U = u_1$, $u_2$, $u_3 = delete(00*)$, $insert(000*, H2)$, $insert(0*, H3)$. For this example, $V(U) = U$ and the incremental consistent orderings are $u_1u_2u_3$, $u_2u_1u_3$, $u_2u_3u_1$, and $u_3u_2u_1$. The remaining two orderings $u_1u_3u_2$ and $u_3u_1u_2$ are not incremental consistent. To see that $u_1u_3u_2$, for example, is not incremental consistent, note that following $u_3$, the next hop for destination addresses that are of the form 000* is $H3$ whereas in the original ordering $u_1u_2u_3$ these destination addresses have next hop $H1$ initially, $H0$ following $u_1$, and $H2$ following both $u_2$ and $u_3$.

Any $H(V)$ that disallows the topological ordering $u_1u_3u_2$ must have at least one of the directed edges $(u_3, u_1)$, $(u_2, u_1)$, and $(u_2, u_3)$. However, the presence of any one of these edges in $H(V)$ also invalidates one of the four permissible orderings. So, no $H(V)$, whose topological orderings coincide exactly with the set of permissible orderings, exists.

We can formulate meta-heuristic (e.g., simulated annealing, genetic, etc.) based algorithms to determine near optimal incremental consistent orderings of $V(U)$. However, given the rather simple relationships that exist in our test update sequences, it is unlikely these will provide incremental orderings that are significantly superior to the batch consistent orderings generated by our algorithm of Figure 8.

# 4 CONSIST

The ideas presented in this paper lead to a system–CONSIST, which optimally reduces the number of operations in an update batch. The system orders the updates in the optimally reduced set so as to guarantee batch or incremental consistency (depending on which type of consistency is desired) when the updates are done one by one in the generated order. Further, the system applies heuristics to minimize the maximum size of an intermediate table. The output of CONSIST may then be provided to any router that supports incremental updates. When this router performs the updates in the provided order, table consistency is maintained.

# 5 Experiments

We have applied our heuristic to obtain near optimal batch consistent sequences on 21 datasets derived from update sequences of various routers [3, 4]. Figure 12 gives the details of these datasets, with the first and second columns showing the name and the total number of prefixes in the initial forwarding table for each dataset. The third column gives the period for which the data has been extracted. Columns four to seven, respectively, give the number of "insert", "delete" and "change" operations, and the total number of operations for each dataset. All the datasets except rrc00Jan25 were collected starting from the zeroth hour on February 1, 2009. The last one, rrc00Jan25, is for the 3 hour period from 5:30am to 8:30am on January 25, 2003, which corresponds to the *SQLSlammer* worm attack [25].

Figure 13 gives the total number of update operations that remain after applying our reduction technique to each update batch. For the column labeled $x = 1$, all updates with the same timestamp define a batch whereas for the column labeled $x = 30$, for example, the updates were partitioned into batches of duration 30 seconds and our reduction method applied to each partition. For the last column in this table, reduction was applied to the entire collection of updates for each data set. So, for example, for rrc00 we applied reduction to the entire set of 447,617 updates. When reduction is applied to batches of updates with the same timestamp, the number of updates decreased between 0.1% (rrc07) and 26.5% (route-views.isc). With a batching interval of 30 seconds, the decrease was between 0.6% (rrc07) and 70.6% (route-views.linx) while with a batching interval of 60 seconds the decrease was between 0.7% (rrc07) and 77.9% (route-views.linx). When reduction is applied to the entire update set, the number of updates decreased between 0.8% (rrc07) and 98.8% (rrc04). Even when reduction is limited to updates with the timestamp, we are able to reduce the total number of updates that need actually be performed by up to 26.5%!

| DataSet | #Prefixes | Collection Period (hours) | *#inserts* | *#deletes* | *#changes* | #Total operations |
|---|---|---|---|---|---|---|
| rrc00 | 294098 | 75.7 | 39553 | 40051 | 368013 | 447617 |
| rrc01 | 276795 | 75.2 | 41692 | 41988 | 492315 | 575995 |
| rrc03 | 283754 | 42.7 | 27702 | 27914 | 292454 | 348070 |
| rrc04 | 288610 | 17 | 16086 | 15977 | 193392 | 225455 |
| rrc05 | 280041 | 103 | 20276 | 18285 | 439647 | 478208 |
| rrc06 | 278744 | 235 | 157549 | 157547 | 289272 | 604368 |
| rrc07 | 275097 | 0.417 | 247 | 218 | 179835 | 180300 |
| rrc10 | 278898 | 105 | 21620 | 22473 | 326720 | 370813 |
| rrc11 | 277166 | 80.2 | 58115 | 58378 | 290621 | 407114 |
| rrc12 | 278499 | 62.3 | 33196 | 33572 | 410464 | 477232 |
| rrc13 | 284986 | 57.8 | 23920 | 23713 | 284710 | 332343 |
| rrc14 | 276170 | 83.6 | 56598 | 56810 | 203955 | 317363 |
| rrc15 | 284047 | 134 | 95790 | 93750 | 183131 | 372671 |
| rrc16 | 282660 | 672 | 3338 | 937 | 8896 | 13171 |
| route-views2 | 294127 | 56.5 | 13882 | 15552 | 679100 | 708534 |
| route-views4 | 275737 | 95 | 69627 | 69754 | 526302 | 665683 |
| route-views.eqix | 275736 | 70.3 | 51104 | 51066 | 253693 | 355863 |
| route-views.isc | 281095 | 68.2 | 44286 | 44444 | 292323 | 381053 |
| route-views.linx | 278196 | 49.1 | 23137 | 23413 | 384344 | 430894 |
| route-views.wide | 283569 | 174 | 101821 | 103862 | 372035 | 577718 |
| rrc00Jan25 | 122801 | 3 | 18275 | 19455 | 272115 | 309845 |

Figure 12: Datasets used in the experiments

Figures 14 and 15 give the number of updates received per second by rrc04 before and after reduction, with the updates being batched every 1, 60, 240, and 480 seconds. We see a sharp spike in Figures 14(a) and (b) around 7000 seconds. This happens due to a sudden withdrawal of a large number of prefixes by router AS12350, presumably due to a failure. This results in a change in nexthop for a large number of prefixes in rrc04. For example, at 6906 seconds (actual timestamp: 02/01/09 12:22:39), there are more than 28000 such changes all happening at the same timestamp. A short while later, probably as AS12350 is back up again, the withdrawn prefixes are announced by AS12350, and the forwarding table in rrc04 is updated again to change the nexthop for those prefixes for which AS12350 is the best nexthop. The two changes from and to AS12350 cancel each other when the batching interval is 480 seconds or more and we thus see a flat plot for Figure 15(h).

After applying reduction to remove the redundant operations, the remaining operations are arranged in a consistent sequence using our near optimal heuristic in Figure 8. Figure 16 gives the growth in the forwarding table compared to the initial table size. The first column gives the names of each dataset whereas the other columns give the maximum increase in size of the intermediate forwarding table as the updates are batched over regular intervals of time as indicated, and are subjected to reduction and consistent sequencing using our heuristic. Thus we see that the maximum growth remains more or less constant, even as updates are batched over extended periods of time, such as, 4 or 8 minutes. When we apply our algorithms to the entire set of updates, then we see a remarkable drop in the maximum growth to zero for most datasets. This can be explained by the availability of more deletes that require no prior inserts, as all the updates are considered. These deletes are put first in the new sequence freeing up enough space in the forwarding table to hold the inserts without any increase in the size compared to the initial forwarding table.

# 6 Conclusion

We have presented a policy for incorporating BGP updates to a router forwarding table in a consistent manner, when a batch of updates is received. By applying the updates consistently to the forwarding table, we ensure that the data packets are forwarded to the proper next hops, avoiding any non-optimal forwarding leading to longer path to destination or packet loss. Two types of consistency, namely, batch consistency and incremental

| DataSet | Before reduction | After applying reduction every $x$ seconds | | | | | | all updates |
|---|---|---|---|---|---|---|---|---|
| | | $x$=1 | $x$=30 | $x$=60 | $x$=120 | $x$=240 | $x$=480 | |
| rrc00 | 447617 | 406386 | 177375 | 123749 | 88800 | 61912 | 47555 | 8626 |
| rrc01 | 575995 | 509778 | 220429 | 160464 | 118095 | 94064 | 71259 | 14156 |
| rrc03 | 348070 | 299810 | 134882 | 99450 | 75445 | 58292 | 45023 | 12898 |
| rrc04 | 225455 | 224838 | 180057 | 144054 | 124833 | 105566 | 8048 | 2609 |
| rrc05 | 478208 | 434369 | 194278 | 145546 | 103576 | 80615 | 59403 | 12682 |
| rrc06 | 604368 | 602107 | 291214 | 201325 | 143512 | 102162 | 73043 | 8361 |
| rrc07 | 180300 | 180093 | 179121 | 178977 | 178864 | 178781 | 178717 | 178717 |
| rrc10 | 370813 | 365548 | 161090 | 130297 | 99840 | 82480 | 67051 | 22254 |
| rrc11 | 407114 | 381224 | 145005 | 104545 | 75211 | 53946 | 43458 | 12049 |
| rrc12 | 477232 | 416304 | 197191 | 160111 | 123181 | 60864 | 48014 | 7662 |
| rrc13 | 332343 | 311341 | 151922 | 125467 | 103022 | 65206 | 48305 | 7482 |
| rrc14 | 317363 | 304862 | 102071 | 77493 | 57973 | 45289 | 34427 | 11860 |
| rrc15 | 372671 | 365096 | 203131 | 163680 | 135709 | 102771 | 86396 | 44262 |
| rrc16 | 13171 | 13053 | 11425 | 11026 | 10711 | 10502 | 10150 | 8935 |
| route-views2 | 708534 | 570930 | 234733 | 165809 | 115778 | 83604 | 62415 | 16012 |
| route-views4 | 665683 | 579070 | 248809 | 172020 | 120384 | 85324 | 61405 | 9447 |
| route-views.eqix | 355863 | 340296 | 133430 | 95903 | 70464 | 50682 | 38131 | 10350 |
| route-views.isc | 381053 | 333569 | 120537 | 86787 | 62237 | 45038 | 32007 | 8198 |
| route-views.linx | 430894 | 316599 | 126638 | 95183 | 74087 | 55147 | 43391 | 12054 |
| route-views.wide | 577718 | 572730 | 242015 | 173544 | 120086 | 88761 | 68421 | 8804 |
| rrc00Jan25 | 309845 | 294656 | 185845 | 140020 | 100616 | 68876 | 46416 | 4763 |

Figure 13: Total number of updates before and after applying reduction

consistency have been introduced as well as a reduction method to identify and remove the redundant updates from a given batch of updates. The insert/delete/change operations to the forwarding table transpiring from the updates arriving in a batch may be conveniently represented as a digraph, with the operations as vertices and edges being introduced between vertices that have predecessor-successor relationship to maintain batch consistency. Any topological ordering of the vertices in the precedence graph gives us a batch consistent sequence. Among all the batch consistent sequences we desire the one that leads to minimum growth of the forwarding table at the time of incorporating the updates. We propose an efficient heuristic that builds a near optimal batch consistent sequence for practical datasets. For incremental consistent orderings, we showed that there is no digraph whose topological orderings correspond to all the permissible incremental consistent orderings.

The contributions of this paper include:

1. The notions of batch and incremental consistency in updates.

2. An algorithm to reduce the number of updates in an update batch to the minimum number while ensuring batch and incremental consistency. Our experiments indicate that a reduction of up to 26.5% is possible when reduction is limited to sets of updates with the same timestamp; larger reduction results when the batching interval for reduction is increased.

3. Algorithms to order the optimally reduced update set so as to ensure batch and incremental consistency when the updates are done one by one.

4. A heuristic to order the reduced update set so as to reduce maximum table size growth when updates are done one by one and while ensuring batch consistency.

The system CONSIST incorporates these contributions to generate near optimal consistent update sequences that may be used by any router that supports incremental updates.

# References


[1] W. Lu and S. Sahni, Low Power TCAMs For Very Large Forwarding Tables, *Proceedings of INFOCOM*, 2008.

[2] W. Lu and S. Sahni, Succinct representation of static packet classifiers, *International Conference on Computer Networking*, 2007.

[3] *http://bgp.potaroo.net*, 2007.

[4] *http://www.ripe.net/projects/ris/rawdata.html*, 2008.

[5] RFC 2439, BGP Route Flap Damping *http://www.rfc-editor.org/rfc/rfc2439.txt*, 1998

[6] RFC 4271, A Border Gateway Protocol 4 (BGP-4), *http://www.ietf.org/rfc/rfc4271.txt*, 2006

[7] S. Sahni, K. Kim, and H. Lu, Data structures for one-dimensional packet classification using most-specific-rule matching, *International Journal on Foundations of Computer Science*, 14, 3, 2003, 337-358.

[8] Z. Wang, H. Che, M. Kumar, and S.K. Das, CoPTUA: Consistent Policy Table Update Algorithm for TCAM without Locking, *IEEE Transactions on Computers*, 53, 12, December 2004, 1602-1614.

[9] F. Zane, G. Narlikar and A. Basu, CoolCAMs: Power-Efficient TCAMs for Forwarding Engines, *INFOCOM*, 2003.

[10] M. Waldvogel, G. Varghese, J. Turner and B. Plattner, Scalable High Speed IP Routing Lookups, *SIGCOMM* 1997

[11] T. Mishra and S.Sahni, PETCAM – A Power Efficient TCAM For Forwarding Tables, *IEEE Symposium on Computers and Communications*, 2009

[12] S. Sikka and G. Varghese, Memory-Efficient State Lookups with Fast Updates, *SIGCOMM* 2000

[13] D. Shah and P. Gupta, Fast Updating Algorithms on TCAMs, *IEEE Micro* Volume 21, Issue 1, Jan-Feb 2001, 36-47

[14] A. Basu and G. Narlikar, Fast Incremental Updates for Pipelined Forwarding Engines, *IEEE/ACM Transactions on Networking* Volume 13, Number 3, June 2005

[15] V. Srinivasan and G. Varghese, Faster IP lookups using controlled prefix expansion, *SIGMETRICS*, 1998.

[16] G. Wang and N. Tzeng TCAM-Based Forwarding Engine with Minimum Independent Prefix Set (MIPS) for Fast Updating, *IEEE International Conference of Communications* Volume 1, June 2006, 103-109

[17] T. Mishra and S.Sahni, DUO-Dual TCAM Architecture for Routing Tables with Incremental Update, *http://www.cise.ufl.edu/~sahni/papers/duo.pdf, To be posted before conference.*

[18] B. Zhang, D. Massey and L. Zhang, Destination Reachability and BGP Convergence Time, *IEEE GLOBECOM* 2004

[19] P. Crescenzi, L. Dardini and R. Grossi, IP Address Lookup Made Fast and Simple, *Lecture Notes in Computer Science* 1643:65-76, 1999

[20] C.Partridge et al., A Fifty Gigabit Per Second IP Router, *IEEE/ACM Transactions on Networking* Volume 6, 237–248, 1998.

[21] Sharad Agarwal, Chen-Nee Chuah, Supratik Bhattacharyya, and Christophe Diot, Impact of BGP Dynamics on Router CPU Utilization, *Passive and Active Network Measurement* 2004

[22] Craig Labovitz, G. Robert Malan, and Farnam Jahanian, Internet Routing Instability, *IEEE Transactions on Networking* Volume 6, Number 5, October 1998

[23] Jun Li, Michael Guidero, Zhen Wu, Eric Purpus, and Toby Ehrenkranz BGP Routing Dynamics Revisited *SIGCOMM Computer Communication Review* Volume 37, Number 2, April 2007

[24] A. Basu and G. Narlikar, Method and apparatus for reducing the number of write operations during route updates in pipelined forwarding engines, *US Patent 7,171,490 B2*

[25] Analysis of BGP Update Surge during Slammer Worm Attack M. Lad, X. Zhao, B. Zhang, D. Massey, and L. Zhang *IWDC* 2003.

[26] K. Zhang, A. Yen, X. Zhao, D. Massey, S. F. Wu and L. Zhang. On detection of anomalous routing dynamics in BGP *Proc. Networking LNCS 3042* 2004.

[27] John Heasley@Routeviews, From an email he sent on 3 March, 2009.

[28] Ola Nordstrom and Constantinos Dovrolis Beware of BGP Attacks *SIGCOMM Computer Communication Review* Volume 34, Issue 2, April 2004.

[29] S. Floyd and V. Jacobson, The synchronization of periodic routing messages, *IEEE/ACM Transaction on Networking* Volume 2, 122–136, April 1994.

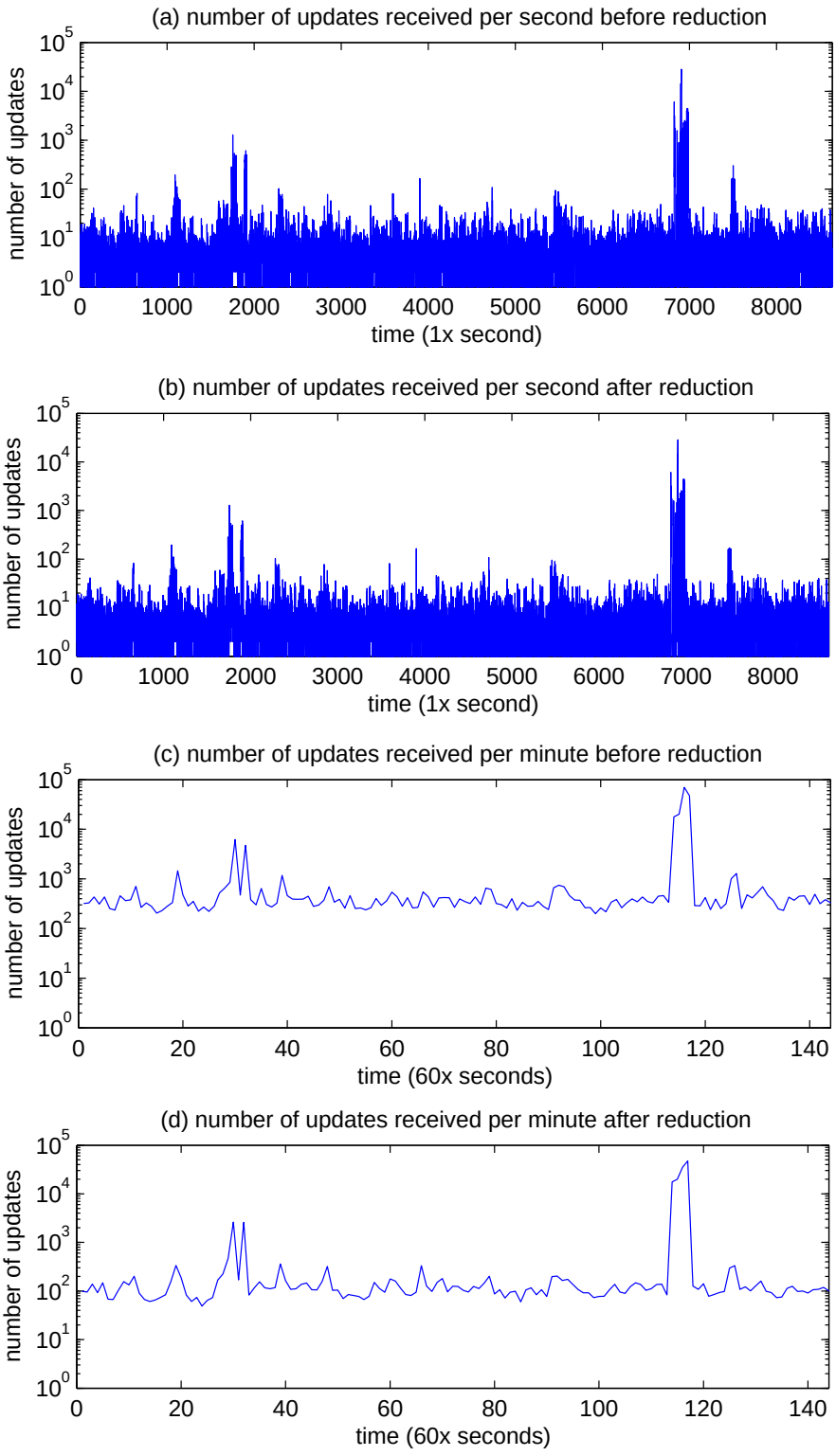



Figure 14: Number of updates before and after reduction (rrc04)

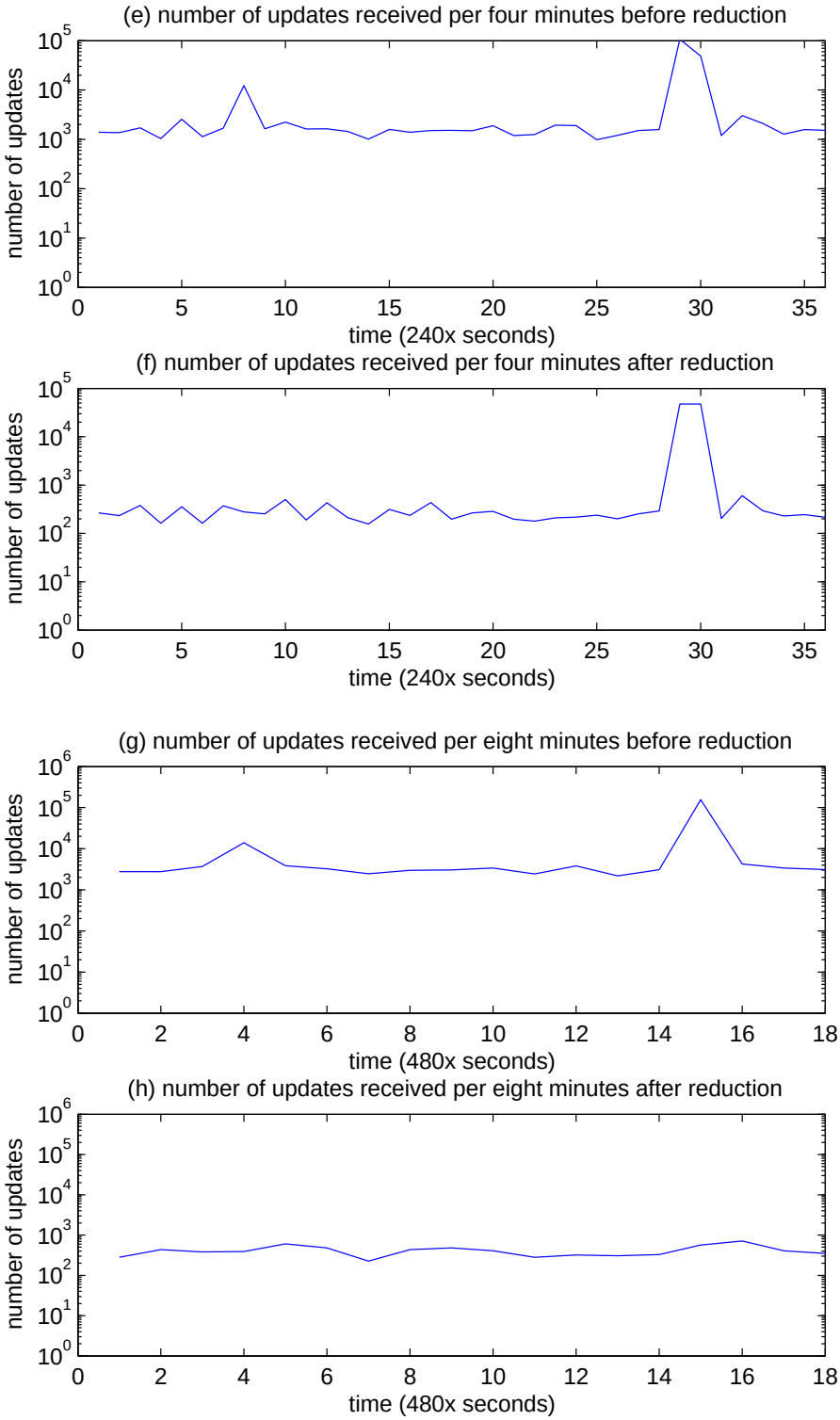



Figure 15: Number of updates before and after reduction (rrc04)

| DataSet | After applying heuristic in Figure 8 every $x$ seconds | | | | | | all updates |
|---|---|---|---|---|---|---|---|
| | $x$=1 | $x$=30 | $x$=60 | $x$=120 | $x$=240 | $x$=480 | |
| rrc00 | 219 | 219 | 219 | 219 | 192 | 192 | 0 |
| rrc01 | 206 | 206 | 206 | 206 | 167 | 167 | 0 |
| rrc03 | 709 | 706 | 700 | 700 | 700 | 700 | 0 |
| rrc04 | 301 | 289 | 289 | 289 | 262 | 256 | 109 |
| rrc05 | 2181 | 2181 | 2181 | 2181 | 2181 | 2177 | 1991 |
| rrc06 | 457 | 457 | 450 | 450 | 450 | 205 | 2 |
| rrc07 | 32 | 29 | 29 | 29 | 29 | 29 | 29 |
| rrc10 | 1259 | 1255 | 1143 | 1143 | 1115 | 1110 | 0 |
| rrc11 | 224 | 196 | 179 | 179 | 179 | 165 | 0 |
| rrc12 | 655 | 653 | 634 | 634 | 634 | 634 | 0 |
| rrc13 | 634 | 632 | 632 | 632 | 531 | 433 | 207 |
| rrc14 | 214 | 175 | 171 | 159 | 158 | 158 | 0 |
| rrc15 | 2420 | 2409 | 2409 | 2409 | 2409 | 2179 | 2040 |
| rrc16 | 2509 | 2506 | 2488 | 2488 | 2401 | 2401 | 2401 |
| route-views2 | 470 | 469 | 439 | 431 | 431 | 431 | 0 |
| route-views4 | 297 | 295 | 295 | 275 | 200 | 192 | 0 |
| route-views.eqix | 493 | 493 | 475 | 463 | 184 | 167 | 38 |
| route-views.isc | 334 | 334 | 319 | 300 | 167 | 167 | 0 |
| route-views.linx | 86 | 86 | 86 | 63 | 63 | 63 | 0 |
| route-views.wide | 995 | 993 | 990 | 990 | 990 | 637 | 0 |
| rrc00Jan25 | 14 | 11 | 11 | 0 | 0 | 0 | 0 |

Figure 16: Maximum increase in intermediate forwarding table size